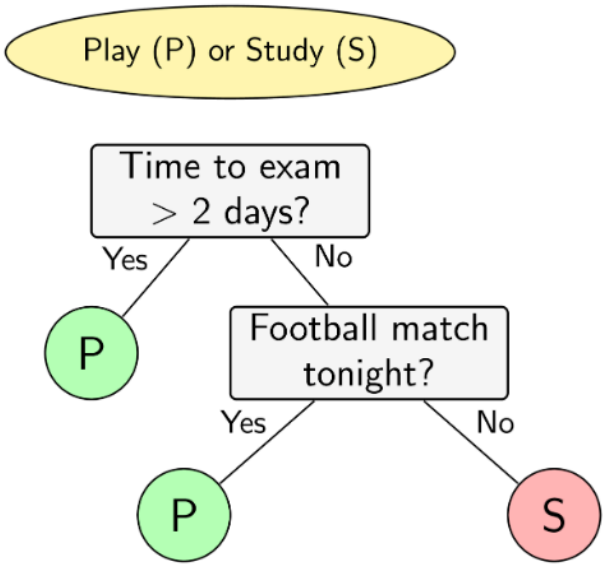
Decision Tree (cây quyết định)

Hoàng Vũ Minh

1. Ý tưởng cơ bản:

Ý tưởng cơ bản của cây quyết định gần giống với if else, nó cũng sẽ đặt ra các điều kiện, các luật sau đó rẽ nhanh để tiến hành phân loại/hồi quy. Cụ thể hơn, nó có cấu trúc như một cây phân cấp, với mỗi nút đại diện cho một quyết định và các nhánh con đại diện cho các kết quả có thể xảy ra của quyết định đó. Cây quyết định được xây dựng bằng cách chia tách dữ liệu thành các tập con nhỏ hơn dựa trên các thuộc tính của dữ liệu, cho đến khi đạt được kết quả mong muốn.

Ví dụ trực quan về cây quyết định:

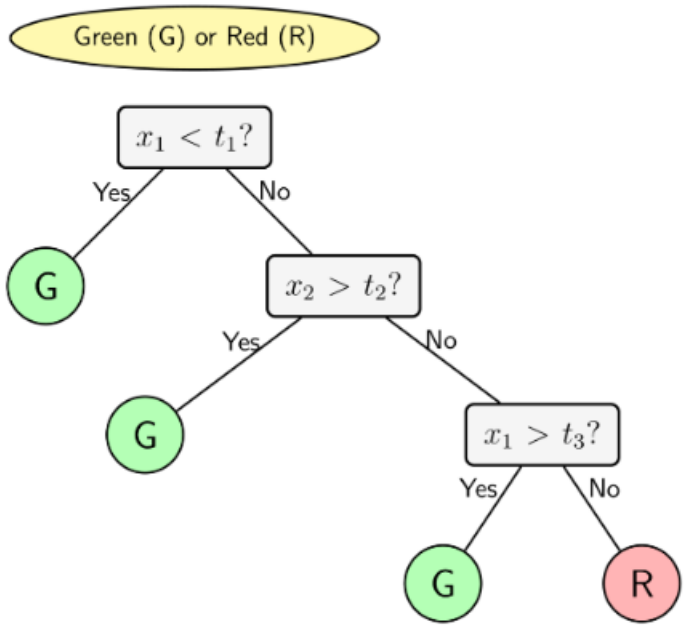


Nếu chỉ dừng ở đây, thoạt nhìn cây quyết định có phần tương đồng với cấu trúc điều kiện if else. Nhưng vấn đề khó ở đây là tìm điều kiện phân chia nào là hiệu quả nhất, là tốt nhất, dựa trên một vài kỹ thuật sẽ được trình bày ở phần sau. Vì vậy, từ đầu vào dữ liệu, nó cũng cần tìm các quy luật sao cho có thể phân tách dữ liệu tốt nhất, đó là lý do tại sao nó lại được liệt vào thuật toán machine learning.

→ Tóm tắt ý tưởng: từ dữ liệu đầu vào, tìm các luật để phân chia dữ liệu sao cho đạt hiệu quả cao nhất. Hiệu quả cao nhất này thường được đánh giá bằng độ tinh khiết của các dữ liệu trong tập con, sẽ được trình bày ở phần sau.

- Ta cần xác định: các luật và thứ tự
- Đo lường: độ tinh khiết

Cấu trúc cây quyết định:



Các ô màu xám, lục hay đỏ đều gọi là các node (trong tiếng việt là nút). Các ô màu xám - các nút điều kiện thì gọi là non-leaf node, các node đỏ lục thể hiện kết quả được gọi là leaf node, node màu vàng gọi là root node.

Trong phần trên ta có nhắc về việc chọn các luật sao cho phân chia tốt nhất, điều này sẽ được trình bày trong thuật toán ID3.

2. *Thuật toán ID3 (Iterative Dichotomiser 3)*

Áp dụng cho bài toán phân loại mà các thuộc tính là categorical, tức là nó có rơi vào hạng mục nào không. Ý tưởng: tại mỗi bước, một thuộc tính tốt nhất sẽ được chọn ra dựa trên một tiêu chuẩn nào đó (chúng ta sẽ bàn sớm). Với mỗi thuộc tính được chọn, ta chia dữ liệu vào các child node tương ứng với các giá trị của thuộc tính đó rồi tiếp tục áp dụng phương pháp này cho mỗi child node.

Tốt nhất ở đây nghĩa là tinh khiết nhất, tức là ta kỳ vọng phép phân chia sẽ tạo ra lớp con sao cho các phần tử trong lớp con đó gần như thuộc cùng một class, gọi là tinh khiết (tức là chỉ có một loại). Nên ta cần một độ đo độ tinh khiết hoặc độ vẩn đục, nhận giá trị thấp nếu tinh khiết và ngược lại. Ta dùng entropy:

$$H(\mathbf{p}) = -\sum_{i=1}^{n} p_i \log(p_i)$$

Cụ thể vào trong thuật toán:
tổng có trọng số của entropy tại các leaf-node sau khi xây dựng decision tree được coi là hàm mất mát của decision tree.
Entropy tại một node:

$$H(\mathcal{S}) = -\sum_{c=1}^{C} \frac{N_c}{N} \log\left(\frac{N_c}{N}\right)$$

Trong đó:

- H(S) là entropy của tập dữ liệu tại nút đang xét
- N là tổng số điểm dữ liệu tại nút đó.
- Nc là số điểm dữ liệu thuộc class c
- C là tổng số lớp

Tổng trọng số tại mỗi node con:

$$H(x, \mathcal{S}) = \sum_{k=1}^{K} \frac{m_k}{N} H(\mathcal{S}_k)$$

Trong đó:

- H(S∣x) là entropy của tập SSS sau khi phân chia dựa trên thuộc tính xxx.
- K là số nút con sau khi phân chia.
- mk là số điểm trong nút con Sk

Information gain: Thông tin thu được khi phân chia tập dữ liệu *S* dựa trên thuộc tính x được định nghĩa là sự khác biệt giữa entropy của tập dữ liệu ban đầu và entropy sau khi phân chia:

$$G(x, \mathcal{S}) = H(\mathcal{S}) - H(x, \mathcal{S})$$

Như vậy tại mỗi node, thuộc tính được chọn sẽ là:

$$x^* = \arg\max_x G(x, \mathcal{S}) = \arg\min_x H(x, \mathcal{S})$$

Nói chung công thức này là để cho gain lớn nhất.

Điều kiện dừng:

- Khi entropy = 0 hoặc nhỏ hơn 1 threshold nào đó.
- Khi độ sâu của cây đạt đến giới hạn. Người ta thường giới hạn độ sâu của cây để giảm overfitting.
- Số phần tử của một node nhỏ hơn một số nào đó
- Tổng số lá nhỏ hơn một số nào đó

Nhìn chung, ta cần điều kiện dừng mục đích là để cây này không bị quá khít với bộ dữ liệu huấn luyện, như vậy có thể khiến chúng ta gặp khó khăn lớn trong các bài toán thực tế, khi dữ liệu là đa dạng.

Kỹ thuật cắt tỉa (Pruning)

Nói chung nó như một kỹ thuật regurlarization hay drop out để tránh overfit:

- Đầu tiên, xây dựng tree tới khi mọi điểm trong training set đều được phân lớp đúng.
- Cách 1: Cắt từ dưới lên, xây xong cây thì cắt từ lá lên trên, rồi thử nghiệm trong validation set, nếu kết quả tốt lên thì lấy.
- Cách 2: thêm regurlazation vào trong hàm loss.

3. Thuật toán CART (Classification and Regression Trees)

Khác với ID3, thuật toán này xây dựng cây quyết định cho cả bài toán hồi quy lẫn phân loại. Với bài toán phân loại thì nhãn dự đoán là categorical

còn với hồi quy thì là một giá trị cụ thể. Nhìn chung thuật toán này gần giống ID3, chỉ là thay entropy bằng Gini impurity

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2$$

Trong đó:

- S là
- pi là tỷ lệ các mẫu thuộc lớp i trong tập con S.
- n là tổng số nhãn

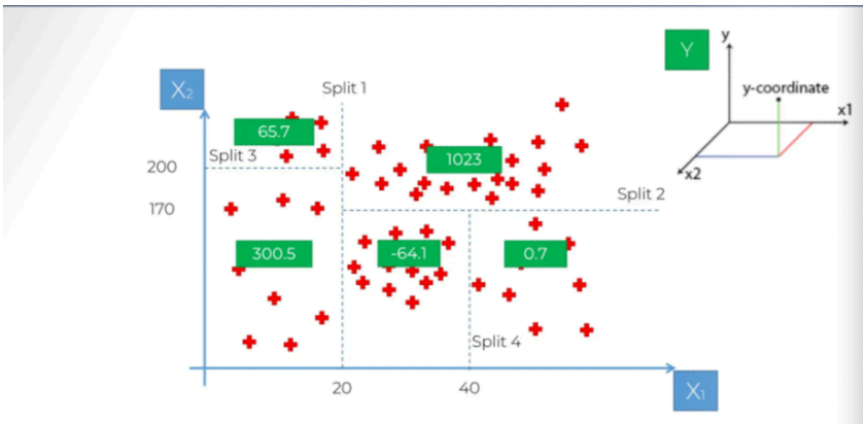
Gini càng thấp thì dữ liệu càng đồng đều, càng tinh khiết.

4. *Nói thêm về cây hồi quy*

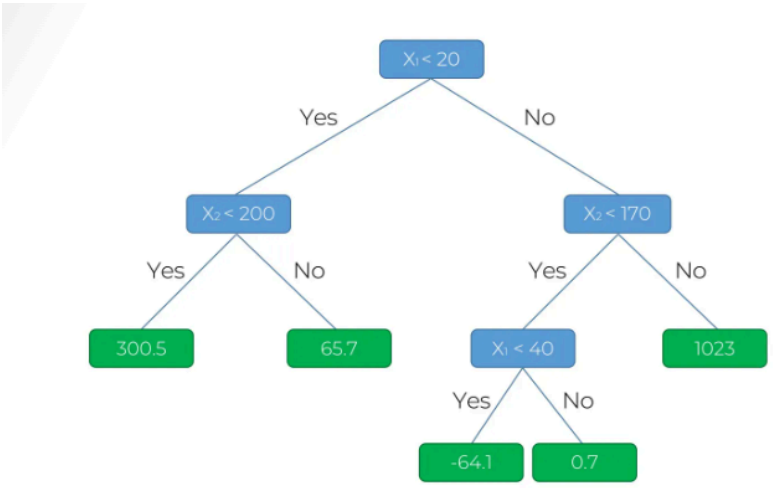
- Nhìn chung, logic của nó cũng khá đơn giản: tìm các điều kiện để chia nhỏ dữ liệu, sau đó nếu điểm dữ liệu mới rơi vào lá nào, giá trị hồi quy dự đoán của nó sẽ bằng trung bình các y trong lá đó.

Ví dụ đơn giản trong hình dưới đây: bộ data của ta có 3 cột x1 x2, y, và trong đó ta đang cần hồi quy y theo x1 và x2, tức là y là biến phụ thuộc, còn x1 và x2 là các biến độc lập.

- Trên hình là không gian 2D nên chỉ biểu diễn x1 và x2, ta coi y là trục thẳng đứng đâm thẳng vào màn hình.
- Các Split 1 2 3 thật ra là các điều kiện chia, và phần màu xanh lá cây là y trung bình của các điểm dữ liệu trong cái lá đó.



Viết dễ hiểu hơn theo cấu trúc cây về các split:



Như vậy, ví dụ ta cần tìm y cho một điểm x1, x2 = (10, 500), bằng cấu trúc cây này ta sẽ có:

- Đi vào node đầu tiên, x1 = 10 <= 20 -> sang nhánh trái.
- x2 = 500 > 200 nên rơi sang nhánh phải (lá phải), trung bình y trong các lá ở đó là 65.7 (màu xanh lá cây).
  ->> giá trị hồi quy là 65.7.

Cây hồi quy cũng có thể dùng thuật toán CART để tìm ra ngưỡng tốt nhất.